Comp. & Maths. with Appls. Vol. 10, No. 2, pp. 123–137, 1984
Printed in the U.S.A.

0097–4943/84 $3.00 + .00

DIAGONALLY IMPLICIT RUNGE–KUTTA FORMULAE FOR THE NUMERICAL INTEGRATION OF NONLINEAR TWO-POINT BOUNDARY VALUE PROBLEMS

J. R. CASH
Department of Mathematics, Imperial College, South Kensington, London SW7 2AZ, England

(*Received October* 1982; *revised April* 1983)

Communicated by Ervin Y. Rodin

Abstract—Fully implicit Runge–Kutta formulae, based on interpolatory quadrature schemes, for the approximate numerical solution of nonlinear two-point boundary value problems have been investigated by Weiss. Such formulae are not used very often in practice, however, because they generally require such a large computational effort as to make them uncompetitive with, for example, integration schemes based on the trapezoidal or implicit mid-point rules. In this paper we consider an alternative class of Runge–Kutta formulae, namely diagonally implicit Runge–Kutta (DIRK) formulae, which can be implemented more efficiently than the fully implicit formulae considered by Weiss. We also consider how these DIRK formulae can be implemented in a defect or deferred correction framework and we give some numerical results to illustrate the algorithms derived. One particular formula belonging to the DIRK class is the implicit mid-point rule. In this paper we derive an efficient implementation of this formula which is applicable when the given boundary conditions are non-separated.

1. INTRODUCTION

In the present paper we shall be concerned with the approximate numerical integration of a class of nonlinear two-point boundary value problems of the form

$$\frac{dy}{dt} = f(t, y(t)), \; a \le t \le b, \tag{1.1a}$$

$$g(y(a), y(b)) = 0, \; y \in \mathscr{R}^N, f: \mathscr{R} \times \mathscr{R}^N \to \mathscr{R}^N. \tag{1.1b}$$

Most two-point boundary value problems of order greater than 1 can be transformed into the form (1.1) although it must be admitted that this is not always the best way to deal with higher order equations.

In an important paper by Weiss[17], which derives much of the theory on which the present paper is based, the numerical solution of (1.1) using a class of fully implicit Runge–Kutta formulae was examined. To derive these formulae consider a grid π_σ:

$$\pi_\sigma = \{t_0, t_1, \ldots, t_\sigma : a = t_0 < t_1 < \ldots < t_\sigma = b\}, \tag{1.2}$$

$$t_i = t_{i-1} + h_{i-1}, \; h = \max_i h_i \le \lambda \min_i h_i,$$

where λ is uniformly bounded for all families of grids to be considered. Let $\{c_i\}_{i=1}^n$ be a set of positive real numbers $\epsilon [0, 1]$ and define $t_{ij} = t_i + c_j h_i$. (Our theory goes through without the c_i having to lie in this range but we impose this restriction in order to avoid the need to evaluate f outside the range $[a, b]$.) One of the classes of fully implicit Runge–Kutta formulae considered by Weiss is given by

$$h_i N_h Y_{ij} \equiv Y_{ij} - Y_i - h_i \sum_{k=1}^{n} a_{jk} f(t_{ik}, Y_{ik}) = 0, \tag{1.3a}$$

$$j = 1, 2, \ldots, n,$$

$$i = 0, 1, \ldots, \sigma - 1,$$

$$Y_{i+1} - Y_i - h_i \sum_{k=1}^{n} b_k f(t_{ik}, Y_{ik}) = 0, \tag{1.3b}$$

$$i = 0, 1, \ldots, \sigma - 1,$$

$$g(Y_0, Y_\sigma) = 0. \tag{1.3c}$$

In [17] it is shown that such schemes are identical to those which can be obtained using collocation with piecewise polynomials. Equations (1.3) define a system of $(n + 1)\sigma + 1$ nonlinear algebraic equations in the unknowns

$$Y_{ij}, j = 1, 2, \ldots, n, i = 0, 1, \ldots, \sigma - 1 \text{ and } Y_i, 0 \leq i \leq \sigma.$$

If these algebraic equations are solved using some form of Newton iteration, the leading term of the operation count to achieve this task is given by Weiss as

$$\sigma[N_1^3/3 + 3N_1^2N/2 + 2N_1N^2]$$

multiplications per iteration where, see [17], $N_1 = N(n + 1 - r)$ if $c_n = 1$ and $N_1 = N(n + 2 - r)$ otherwise with $r = 1$ if $c_1 > 0$ and $r = 2$ otherwise. This is in contrast to schemes based on the trapezoidal or implicit mid-point rules where the leading term in the operation count to solve the algebraic equations resulting from a Newton iteration is $7\sigma N^3/3$ multiplications[13]. (In this paper we shall give an algorithm for the implicit mid-point rule which reduces this leading term to $4\sigma N^3/3$ multiplications.

Throughout this paper we shall assume that N is sufficiently large so that the leading terms in the operation counts required to perform the linear algebra (i.e. those terms proportional to σN^3) dominate the other terms in the operation count. When this is the case, the relative costs of the linear algebra for two different discretization schemes can be compared by examining the leading term in their operation counts. It is well known that schemes based on linear multistep methods, such as the deferred correction approach of Lentini and Pereyra for example[13], are normally considerably more efficient than those based on fully implicit Runge–Kutta formulae. In the case where the boundary conditions are separated, the linear algebra cost associated with these two classes of formulae can be reduced considerably [12, 17] but the general remarks regarding their relative costs still apply. We shall return to the question of how large we require N to be in practice in Section 3.

Of course, the high computational effort associated with fully implicit Runge–Kutta formulae has been known for some time. Two significant attempts to overcome this difficulty have been made for initial value problems. These are the transformation methods of Butcher[3], Bickart[2] and Varah[16] and the diagonally implicit Runge–Kutta (DIRK) methods first considered by Nørsett[1, 14, 5, 6]. While it is not clear that we can extend the transformation methods to deal efficiently with the solution of two-point boundary value problems, such an extension is possible with DIRK formulae. The purpose of the present paper is to describe a method for the numerical integration of (1.1) using DIRK formulae and to give an efficient algorithm for the solution of the resulting system of nonlinear algebraic equations. We also examine methods for improving the order of accuracy of the underlying DIRK formula and give some numerical results to illustrate the methods derived.

Finally in this section we attempt to define more clearly the class of problems (1.1) with which we shall be concerned. Such a classification is difficult and necessarily imprecise but we feel that some remarks at this stage would be valuable. As mentioned previously, the problems which concern us are those which are such that the cost of the linear algebra dominates the total computational cost and where N is sufficiently large so that terms proportional to σN^3 dominate the total linear algebra cost. In addition to having this property, the problem to be solved should also have one or more of the following

properties:

(1) Low or "engineering" precision is required. It is difficult to quantify this requirement but for some additional comments see[7].

(2) Storage limitations demand that only a "few" grid points be used to obtain the solution. A situation where this problem may arise is, for example, in the method of lines solution of partial differential equations and exactly how many grid points are allowed will depend on the machine being used for computing. In such cases there may not be sufficient grid points to allow the computation of all the corrections necessary to obtain the solution efficiently by deferred correction. Here we would expect a Runge–Kutta approach, which is one step in nature, to be valuable.

(3) The boundary conditions are non-separated since it is in this situation that we can offer the largest savings over conventional methods.

2. DIAGONALLY IMPLICIT RUNGE-KUTTA FORMULAE

The general n stage Runge–Kutta formula for the numerical integration of (1.1a) can be written as

$$Y_{i+1} - Y_i = h_i \sum_{k=1}^{n} b_k f(t_i + c_k h_i, Y_{ik}), \tag{2.1a}$$

$$Y_{ij} = Y_i + h_i \sum_{k=1}^{s} a_{jk} f(t_i + c_k h_i, Y_{ik}), j = 1, 2, \ldots, n; s \leq n. \tag{2.1b}$$

For a fully implicit Runge–Kutta formula we have $s = n$ and, as was pointed out in the previous section, the cost of the linear algebra associated with these formulae, compared with linear multistep methods, is relatively high and becomes increasingly so as n increases. If, however, we put $s = j$ and $a_{jj} = \alpha$ for $j \in [1, n]$ we obtain a diagonally implicit Runge–Kutta (DIRK) method. Such methods are efficient for the numerical integration of stiff initial value problems and the main aim of the present paper is to show that this efficiency also carries over to boundary value problems.

Consider now the application of the diagonally implicit Runge–Kutta formula

$$h_i N_h Y_{ij} \equiv Y_{ij} - Y_i - h_i \sum_{k=1}^{j-1} a_{jk} f(t_{ik}, Y_{ik}) - h_i \alpha f(t_{ij}, Y_{ij}) = 0,$$

$$1 \leq j \leq n, \tag{2.2}$$

to (1.1). We shall assume that $c_1 > 0$, $c_n < 1$ although our approach can easily be extended to include the case $c_1 = 0$ and/or $c_n = 1$. Together with (2.2) we have the additional relations

$$E_h(Y_i) \equiv Y_{i+1} - Y_i - h_i \sum_{k=1}^{n} b_k f(t_{ik}, Y_{ik}) = 0. \tag{2.3}$$

and

$$g(Y_0, Y_\sigma) = 0. \tag{2.4}$$

Equations (2.2)–(2.4) define $\sigma(n+1)+1$ relations for the $\sigma(n+1)+1$ unknowns Y_i, Y_{ij}, $j = 1, 2, \ldots, n$; $i = 0, 1, \ldots, \sigma - 1$ and Y_σ. However, because of the fact that the Runge–Kutta formulae we are using have a special "triangular form" ($a_{ij} = 0$ for $j > i$, $a_{jj} = \alpha$ for all j), the algebraic equations to be solved also have a triangular form. It is due entirely to the fact that the algebraic equations have this sparseness property, rather than being dense as they are with the fully implicit Runge–Kutta formulae considered by Weiss, that we are able to find an efficient algorithm for their solution. The nonlinear system of

algebraic equations to be solved can now be written as

$$\Phi(Y) \equiv \begin{bmatrix} g(Y_0, Y_\sigma) \\ N_h Y_{01} \\ \vdots \\ N_h Y_{0n} \\ E_h(Y_0) \\ \vdots \\ N_h Y_{\sigma-1n} \\ E_h(Y_{\sigma-1}) \end{bmatrix} = 0 \tag{2.5}$$

where $Y \equiv (Y_0, Y_{01}, \ldots, Y_{0n}, Y_1, Y_{11}, \ldots, Y_{1n}, \ldots, Y_\sigma)^T$.

If a solution of this system is obtained using some form of Newton iteration we shall be faced with the task of solving a linear algebraic system, for successive iterates Y^ν to Y, having the form

$$Q^\nu[Y^{\nu+1} - Y^\nu] = -\Phi(Y^\nu), \nu = 0, 1, \ldots, \tag{2.6}$$

where the matrix Q^ν has a special sparseness structure. To illustrate this structure we consider the case $\sigma = 3$. Here the form taken by Q^ν is

$$\begin{bmatrix} G_1 & & & G_2 \\ A_1 & & & \\ & A_2 & & \\ & & A_3 & \end{bmatrix} \tag{2.7}$$

where the A_i are of size $(n+1)N \times (n+2)N$ and the G_i are of size $N \times N$. To solve (2.5) we shall use the modified Newton iteration scheme obtained by setting

$$\frac{\partial f}{\partial y}(t_i + c_k h_i, Y_{ik}) = \frac{\partial f}{\partial y}(t_i, Y_i) = J_i \text{ say.}$$

In this case the precise form taken by each of the blocks A_i is

$$\begin{bmatrix} -I & I - h\alpha J_i & 0 & 0 & \cdots & 0 \\ -I & -ha_{21}J_i & I - h\alpha J_i & 0 & \cdots & 0 \\ \vdots & & & & & \\ -I & -ha_{n1}J_i & -ha_{n2}J_i & \cdots & I - h\alpha J_i & 0 \\ -I & -hb_1 J_i & -hb_2 J_i & \cdots & -hb_n J_i & I \end{bmatrix} \tag{2.8}$$

Since each of these blocks is in "almost lower triangular" form, it is plausible that we can find an efficient algorithm for the solution of the algebraic system (2.6). In the next section

we shall describe such an algorithm and given the operation count required for the solution.

3. EFFICIENT SOLUTION OF THE ALGEBRAIC EQUATIONS

3.1 *Non-separated boundary conditions*

A simple example—the implicit mid-point rule. To start off this section we consider the numerical integration of (1.1) using the implicit mid-point rule

$$Y_{i+1} - Y_i = h_i f\left(t_i + \frac{1}{2}h_i, \frac{Y_i + Y_{i+1}}{2}\right). \tag{3.1}$$

An examination of the mid-point rule is relevant for two important reasons. Firstly, it makes the algorithm for solving the linear algebraic equations transparent and allows the general algorithm to be understood more easily. Secondly, the implicit mid-point rule is, of course, an important integration method in its own right. Keller[12] has studied this scheme in detail and has shown that it is easier to use than the trapezoidal rule in the case of piecewise continuous data and Keller has also reported that the implicit mid-point rule is "frequently more efficient" than the trapezoidal rule. Furthermore, for linear problems, the trapezoidal and implicit midpoint rules are equivalent. In view of this any improvement in the implementation of the implicit mid-point rule will carry over to algorithms, such as the important deferred correction algorithm of Lentini and Pereyra[13], based on the trapezoidal rule when solving linear problems.

Applying (3.1) to the numerical solution of (1.1) and solving the resulting system of nonlinear algebraic equations using a Newton iteration scheme, we are required at each iteration to solve a system of the form (2.6) where the coefficient matrix has the form

$$\begin{bmatrix} G_1 & & & & G_\sigma \\ S_1 & R_1 & & & \\ & S_2 & R_2 & & \\ & & \ddots & \ddots & \\ & & & S_\sigma & R_\sigma \end{bmatrix}.$$

Lentini and Pereyra[13] have given an algorithm for the solution of this linear system of equations and have shown that the leading term in the operation count for their algorithm is $\frac{7}{3}\sigma N^3$ multiplications. In the first part of this section we demonstrate the remarkable fact that if the implicit mid-point rule is rewritten as a diagonally implicit Runge–Kutta method, this operation count can be reduced to $\frac{4}{3}\sigma N^3$ multiplications. In the second part of this section we extend our algorithm to arbitrary DIRK formulae and compare the operation count to that of certain other methods.

Rewriting the implicit mid-point rule as a DIRK formula, we have

$$Y_{i1} = \frac{Y_{i+1} + Y_i}{2} = Y_i + \frac{h_i}{2} f(t_{i+\frac{1}{2}}, Y_{i1}), \tag{3.3a}$$

$$Y_{i+1} - Y_i = h_i f(t_{i+\frac{1}{2}}, Y_{i1}). \tag{3.3b}$$

Instead of taking the unknowns as being only the $\{Y_i\}_{i=0}^{\sigma}$, as with conventional implementations of the implicit mid-point rule, we take the unknowns in each sub-interval $[t_i, t_{i+1}]$ as being Y_i, Y_{i1}, Y_{i+1}. The linear system of algebraic equations to be solved has

the coefficient matrix

$$\begin{bmatrix} G_1 & & & & & & G_\sigma \\ -I & I-\frac{1}{2}hJ_1 & & & & & \\ I & -2I & I & & & & \\ & & -I & I-\frac{1}{2}hJ_2 & & & \\ & & I & -2I & I & & \\ & & & & \ddots & \ddots & \\ & & & & I & -2I & I \end{bmatrix}. \tag{3.4}$$

To obtain the solution of this linear system of algebraic equations we use exactly the algorithm suggested by Lentini and Pereyra[13]. This involves writing (3.4) in the partitioned form

$$\begin{matrix} N\{ \\ 2\sigma N\{ \\ \\ \underbrace{}_{N} \end{matrix} \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \begin{matrix} \\ \\ \\ \underbrace{}_{2\sigma N} \end{matrix}. \tag{3.5}$$

The linear system of equations to be solved now has the form

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x_0 \\ \hat{x} \end{bmatrix} = \begin{bmatrix} b_0 \\ \hat{b} \end{bmatrix}. \tag{3.6}$$

Following Lentini and Pereyra[13] we put $V = D^{-1}C$, $w = D^{-1}\hat{b}$ and define

$$\tilde{C} = [C/\hat{b}],\ \tilde{V} = [V/w].$$

The solution of (3.6) is

$$x_0 = (A - BV)^{-1}(b_0 - Bw), \tag{3.7a}$$

$$\hat{x} = D^{-1}(\hat{b} - Cx_0). \tag{3.7b}$$

It remains to compute V and w and these are obtained by solving the system

$$D\tilde{V} = \tilde{C}. \tag{3.8}$$

Because of the special structure of the matrix D, this system can be solved very cheaply. Putting

$$\tilde{V} = \begin{bmatrix} V_{11} \\ V_{12} \\ V_{21} \\ V_{22} \\ \vdots \\ V_{\sigma 2} \end{bmatrix}$$

where each of the V_{ij} is of size $N \times (N+1)$, the V_{ij} satisfy the two recurrences

$$[I - \tfrac{1}{2}h_i J_i]V_{i1} = V_{i-12} + \tilde{C}_{i1},\ i = 1, 2, \ldots, \sigma;\ V_{02} = 0 \tag{3.9a}$$

$$V_{i2} = 2V_{i1} - V_{i-12} + \tilde{C}_{i2}, \tag{3.9b}$$

where $\tilde{C}_{i1}$, $\tilde{C}_{i2}$ denote appropriate partitions of the matrix $\tilde{C}$. These recurrences can be solved for the V_{ij} using a total of $\sigma(\frac{4}{3}N^3 + N^2)$ multiplications. The calculation of (3.7a), which entails one matrix–matrix product, one matrix vector product, and the solution of one system of linear algebraic equations, requires $\frac{4}{3}N^3$ multiplications. Finally, to solve (3.7b) for $\hat{x}$ we require $2N^2\sigma$ multiplications. Thus the total number of multiplications (most significant terms only) is $\frac{4}{3}\sigma N^3$ compared with $\frac{7}{3}\sigma N^3$ for the implementation of the implicit mid-point rule given in [13]. We again emphasize that for linear problems (for such problems the trapezoidal and implicit mid-point rules are the same) this saving is also obtained for the trapezoidal rule.

The algorithm for a general n-stage DIRK formula goes through in a similar way. The system is first partitioned into the form (3.6) where now D is $(n+1)\sigma N \times (n+1)\sigma N$ with similar changes in the dimensions of B and C. The precise structure of the algebraic equations to be solved is defined by (2.7) and (2.8). However, before computing the solution of this algebraic system we first carry out some pre-conditioning on the A_i appearing in (2.7). The first step of this is to operate on (2.8) by taking

$$\text{the } (k+1)\text{st row} - \text{1st row} \times \frac{a_{k+11}}{\alpha},\ k = 1, 2, \ldots, n-1;$$

$$\text{last row} - \text{1st row} \times \frac{b_1}{\alpha}$$

for each A_i. This reduces A_i to the form

$$\begin{bmatrix} d_{10}I & I - h\alpha J_i & & & \\ d_{20}I & d_{21}I & I - h\alpha J_i & & \\ d_{30}I & d_{31}I & -ha_{32}I & I - h\alpha J_i & \\ \vdots & & & & \\ d_{n+10}I & d_{n+11}I & -hb_2 J_i & \cdots\cdots\cdots & -hb_n J_i \quad I \end{bmatrix} \tag{3.10}$$

where

$$\left.\begin{aligned} d_{j0} &= -1 + \frac{a_{j1}}{\alpha}, \\ d_{j1} &= -\frac{a_{j1}}{\alpha}, \end{aligned}\right\}\ j = 2, 3, \ldots, n$$

$$d_{n+10} = -1 + b_1/\alpha,$$

$$d_{n+11} = -b_1/\alpha,$$

$$d_{10}I = -1.$$

Continuing this process by taking

$$\text{row}(j) - \text{row}(k) \times \frac{a_{jk}}{\alpha} \quad k = 2, 3, \ldots, n-1;\ j = k+1, k+2, \ldots, n,$$

$$\text{row}(n+1) - \text{row}(j) \times \frac{b_j}{\alpha},\ j = 2, 3, \ldots, n.$$

each A_i is transformed into

$$\begin{bmatrix} d_{10}I & I-h\alpha J_i & & & & \\ d_{20}I & d_{21}I & I-h\alpha J_i & & & \\ d_{30}I & d_{31}I & d_{32}I & I-h\alpha J_i & & \\ \vdots & & & & & \\ d_{n0}I & d_{n1}I & d_{n2}I & \cdots\cdots & d_{nn-1}I & I-h\alpha J_i \\ d_{n+10}I & d_{n+11}I & \cdots\cdots & & d_{n+1n}I & I \end{bmatrix}. \tag{3.11}$$

This pre-conditioning requires a total of $\frac{1}{2}n(n+1)N$ multiplications. We now use an extension of the algorithm just defined to solve equations (3.8). Defining

$$V = \begin{bmatrix} V_{11} \\ V_{12} \\ \vdots \\ V_{1n+1} \\ \vdots \\ V_{\sigma n+1} \end{bmatrix}$$

where each V_{ij} is of size $N \times N$, the recurrences defining the V_{ij} are

$$[I-\alpha hJ_i]V_{i1} = V_{i-1n+1} + \tilde{C}_{i1} \quad i = 1, 2, \ldots, \sigma, \; V_{0n+1} = 0. \tag{3.12a}$$

$$[I-\alpha hJ_i]V_{ij} = -\sum_{m=1}^{j-1} d_{jm}V_{im} + \tilde{C}_{ij} - d_{j0}V_{i-1n+1}, j = 2, 3, \ldots, n. \tag{3.12b}$$

$$V_{in+1} = -\sum_{m=1}^{n} d_{n+1m}V_{im} + \tilde{C}_{in+1} - d_{n+10}V_{i-1n+1}. \tag{3.12c}$$

This system of recurrences can be solved using a total of (most significant terms only $((\frac{1}{3}+n)N^3 + (n^2/2)N^2)$ multiplications. Adding in the contribution from the pre-conditioning, the total number of multiplications required is

$$\sigma((\tfrac{1}{3}+n)N^3 + \tfrac{1}{2}n^2N^2 + \tfrac{1}{2}n^2N) + 2\sigma nN^2. \tag{3.13}$$

There are now some general remarks that we wish to make about these algorithms and their operation counts. First we note that equations (3.9a, b) can be reduced to the more familiar form by taking (3.9a) $-$ $(\frac{1}{2}I - \frac{1}{4}hJ_i) \times$ (3.9b). Secondly, we note that our algorithm, unlike many others, does not offer any particular advantage to separated boundary conditions. (However, we shall consider an alternative algorithm for separated boundary conditions in the second half of this section.) If we compare this implementation of the implicit mid-point rule with the one given by Lentini and Pereyra[13] it is easy to see that our formulation saves one matrix–matrix multiplication, thus accounting for the saving of σN^3 multiplications. Thus, in the case of non-separated boundary conditions our implementation offers a significant advantage over other implementations. For more general DIRK formulae the leading term in the operation count is given by (3.13). We see that if $n \geq 2N$ the largest term in (3.13) will be $\frac{1}{2}n^2N^2$ and the leading term in the operation count will be much greater than that for Lentini and Pereyra's algorithm. In view of this, when using DIRK formulae, we shall normally require $n < N$ and this is what we mean when we refer to "large N" in Section 1.

3.2 *Separated boundary conditions*

Existing algorithms for solving (1.1) with separated boundary conditions normally express the coefficient matrix of the algebraic system to be solved in block tri-diagonal form and are thus able to obtain the required solution in something less than σN^3 multiplications[12]. In what follows we shall give an algorithm which is specially designed for separated boundary conditions and, as before, we start off by considering the implicit mid-point rule. In the case of separated boundary conditions our algorithm in general offers little advantage over existing algorithms for the implicit mid-point rule and we consider this case mainly to clarify the general algorithm.

The system of algebraic equations which concerns us is again of the general form (3.4) but where G_1, G_σ now have the sparseness patterns

$$G_1 = \begin{matrix} p \\ q \end{matrix}\begin{bmatrix} x & x & \cdots & x \\ x & & & x \\ x & & & x \\ 0 & & & 0 \\ 0 & & & 0 \\ 0 & & & 0 \end{bmatrix}, \quad G_\sigma = \begin{matrix} \\ p \\ \\ \\ q \\ \end{matrix}\begin{bmatrix} 0 & & & 0 \\ 0 & & & 0 \\ x & & & x \\ x & & & x \end{bmatrix}, \quad p + q = N.$$

We now factorise (3.4) into a UL product in the following way

$$\begin{bmatrix} U_{11} & U_{12} & \cdots & U_{12\sigma+1} \\ I & 0 & \cdots & 0 \\ & I & \ddots & 0 \\ & & \ddots & \vdots \\ & & & I \end{bmatrix} \begin{bmatrix} I - h\alpha J_1 & & & & & \\ -I & I - h\alpha J_1 & & & & \\ I & -2I & I & & & \\ & & -I & I - h\alpha J_2 & & \\ & & I & -2I & I & \\ & & & \ddots & \ddots & \ddots \\ & & & & -I & I - h\alpha J_\sigma \\ & & & & I & -2I & I \end{bmatrix} \tag{3.14}$$

Solving for the $N \times N$ matrices $U_{1j}, j = 1, 2, \ldots, 2\sigma + 1$, we have

$$\left.\begin{aligned} U_{1\,2j}(I - h\alpha J_j) &= 2U_{12j+1} \\ U_{1\,2j-1} &= U_{12j} - U_{12j+1} \end{aligned}\right\} \quad j = \sigma, \sigma - 1, \ldots, 2.$$

$$\begin{aligned} U_{1\,2\sigma+1} &= G_\sigma \\ U_{1\,2}(I - h\alpha J_1) &= 2U_{13} \\ U_{1\,1}(I - h\alpha J_1) &= U_{12} - U_{13} + G_1 \end{aligned} \tag{3.15}$$

The main computational effort required by this algorithm comes from

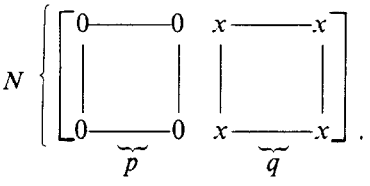
(1) LU factorising the σ matrices $I - h\alpha J_i$, $i = 1, 2, \ldots, \sigma$, into a product $L_i U_i$.

(2) Solving systems of equations of the form $U_{12j}(I - h\alpha J_j) = 2U_{12j+1}$.

For (1) the total operation count is (leading term only) $\sigma N^3/3$ multiplications. For (2) we have $U_{1\,2j}(L_j U_j) = 2U_{1\,2j+1}$. So

$$U_j^T L_j^T U_{12j}^T = 2U_1^T{}_{2j+1}. \tag{3.16}$$

However, each $U^T_{1\,2j+1}$ has the special structure



Clearly U^T_{12j} also has this special structure. In view of this we need only solve for the last q columns of U^T_{12j} and this requires qN^2 multiplications. The total computational effort required to implement this algorithm is (leading terms only)

$$\sigma(\tfrac{1}{3}N^3 + (1+q)N^2)$$

multiplications and this gives a small saving in computational effort over conventional methods for solving the separated boundary conditions case.

The algorithm for general n stage DIRK formulae follows very similar lines. However, in this case the matrices L and U are both of size $((n+1)\sigma+1)N \times ((n+1)\sigma+1)N$. The UL decomposition follows similar lines to before with the unknown matrices in the top row of U being U_{1j}, $1 \le j \le n\sigma + 1$. This UL decomposition requires

$$\sigma\left(\frac{1}{3}N^3 + nqN^2 + \frac{n(n+1)}{2}qN\right) \text{multiplications,}$$

and the backsolves require a further σnN^2 multiplications. As was mentioned in the first part of this section, these algorithms will normally only be used in the case $n < N$.

Finally we note that the operation counts for the general n-stage DIRK formulae are generalisations of the corresponding operation counts for the implicit mid-point rule since the mid-point rule is simply a one-stage DIRK formula.

4. THE FORMULAE

Diagonally implicit Runge–Kutta formulae have been widely used for the numerical solution of stiff initial value problems and many such formulae appear in the literature (see e.g. [6, 9, 14]). The formulae which we shall use in this paper are

Order 2

$$\begin{array}{c|cc} \frac{1}{2} & \frac{1}{2} & \\ \hline & 1 & \end{array} \tag{4.1}$$

Order 4 (Nørsett[14])

$$\begin{array}{c|ccc} \lambda & \lambda & & \\ \frac{1}{2} & \frac{1}{2}-\lambda & \lambda & \\ 1-\lambda & 2\lambda & 1-4\lambda & \lambda \\ \hline & b_1 & b_2 & b_3 \end{array} \tag{4.2}$$

where $b_1 = b_3 = 1/(6(1-2\lambda)^2)$, $b_2 = 1 - 2b_1$ and where λ is one of the three roots of the cubic $\lambda^3 - 3\lambda^2/2 + \lambda/2 - 1/24 = 0$. Here we have used the matrix method of writing Runge–Kutta formulae as defined by Butcher[4]. The three roots of the cubic are approximately $\lambda_1 = 1.06858$, $\lambda_2 = 0.30254$, $\lambda_3 = 0.12889$. Since $\lambda = \lambda_1$ gives the best stability properties for initial value problems it is this value that we shall use in our numerical experiments.

Diagonally implicit Runge–Kutta formulae are normally most effective for the solution

of two-point boundary value problems when they are used together with a method for improving the order of accuracy of the solution obtained. In this paper we mention three methods for increasing accuracy, namely iterated defect correction, extrapolation and deferred correction. All of these methods have the property that they do not require an asymptotic expansion for the global discretization error of the formula being used to be known. This is a distinct advantage when using Runge–Kutta formulae since the expressions for their discretization errors are normally very complicated.

4.1 *Iterated defect correction*

The theory behind the use of iterated defect correction as applied to one step Runge–Kutta methods for the solution of two-point boundary value problems is now well established (see [10, 18]). Note, however, that both of these references are concerned with the solution of the special second order equation $y'' = f(x, y)$ although the theory extends immediately to first order systems. Since, after one application of a defect correction algorithm used with a formula of order p, the order is increased from p to $2p$ (providing the degree of the interpolating polynomial used to define the neighbouring problem is sufficiently high) this approach will be at its most powerful when the integer p is "reasonably large". One of the main contributions of the present paper is to describe efficient high order one step integration schemes which are ideal for use with a defect correction algorithm and we shall illustrate the use of (4.1) and (4.2) in the next section.

4.2 *Deferred correction*

A very general theory justifying the use of deferred correction and proving order of accuracy results has been given by Skeel [15]. In [7] a special class of Runge–Kutta formulae was implemented based on the ideas of Skeel. However, for non-separated boundary conditions the linear algebra cost associated with the methods of [7] is considerably higher than that for the implicit mid-point rule. DIRK formulae fit immediately into the general framework of [7] and the approach discussed there can be used to increase the order of accuracy of our formulae. A numerical example of this is given in Section 5.

4.3 *Richardson extrapolation*

Although extrapolation methods have recently, with good justification, become less popular than ones based on deferred correction, the results of the previous section indicate that, for non-separated boundary conditions, the implicit mid-point rule with extrapolation can be competitive with deferred correction methods. This is particularly so when low accuracy solutions are sought and so only a low order discretization method is required. For example, if the grid π_σ of (1.2) is used with an equally spaced interval h and I_1 iterations are required by the Newton scheme to satisfy some convergence criterion, the computational effort required to solve the algebraic equations is

$$\tfrac{4}{3} I_1 \sigma N^3$$

multiplications (again we have given just the leading term). If we also compute a solution on meshes with grid spacing $h/2$, $h/3$, and the number of Newton iterates required to achieve convergence is respectively I_2, I_3, the total computational effort required to solve the algebraic equations is

$$\tfrac{4}{3} \sigma N^3 (I_1 + 2I_2 + 3I_3)$$

multiplications. This is in contrast to the trapezoidal rule applied with a grid spacing $h/3$ which would require $7\sigma N^3 T_3$ multiplications if the Newton scheme converges in T_3 iterations. Normally we would expect $I_3 = T_3$ and if we assume $I_1 = I_2 = I_3$ this operation count is $8\sigma N^3 I_3$ versus $7\sigma N^3 I_3$ multiplications. However, since the solution obtained on a grid $h/3$ will normally give a very good approximation to the solutions computed with grid spacing $h/2$ and h, we would expect I_1 and I_2 to be less than I_3 if the problem is nonlinear. We would then expect the difference in computational effort to solve the algebraic

equations to be quite modest and, with the information available, we will be able to obtain fourth order and sixth order approximations to the solution using extrapolation.

5. NUMERICAL RESULTS

In this section we shall demonstrate some of the algorithms discussed in this paper by giving some numerical results. When applying the iterated defect correction algorithm it is necessary to define a neighbouring perturbed problem (see [18]). All problems considered have the vector form

$$y' = f(t, y), y \in \mathscr{R}^N$$

and we assume that we have computed an approximate solution

$$\eta^{(0)} = (\eta_0^{(0)}, \eta_1^{(0)}, \ldots, \eta_\sigma^{(0)})^T$$

where $\eta_i^{(0)} \in \mathscr{R}^N$, $\forall_i$. In all our numerical experiments we have chosen the perturbed problem to have as its kth component

$$y_k' = f_k(t, y) + P_k'(t) - f_k(t, P(t)) \text{ for all } k \epsilon [1, N]$$

where $P_k(t)$ is the polynomial interpolating the kth component of the first iterate $\eta^{(0)}$ and $P(t)$ is the vector $(P_1(t), P_2(t), \ldots, P_N(t))$.

The first problem we choose is

Problem 1

$$y'' = e^y, y(0) = y(1) = 0.$$

We choose this because it has been considered as a test problem by many authors and allows us to make comparisons with several other methods. We use a grid spacing of $h = \frac{1}{4}$ and define our interpolating polynomial as the Hermite interpolant of degree 9 interpolating $(t_i, \eta_i^{(0)})$, $(t_i, f(t_i, \eta_i^{(0)}))$, $i = 0, 1, 2, 3, 4$. To obtain a solution the test equation was reduced to a first order system as

$$y' = z \quad y(0) = 0$$
$$z' = e^y, \quad y(1) = 0.$$

In Table 1 we give the results comparing the fourth order formula (4.2) with certain related methods. We note that our algorithm compares well with the 6th order Runge–Kutta formula of Weiss and the extrapolation scheme of Keller. These latter two methods both have a very high cost for the linear algebra and, in particular, the fully implicit

Table 1. Performance on problem 1

Algorithm	Max|Error in y|	Max|Error in z(0)|
4th Order Defect Correction (h = 1/4)		
0 corrections	$.409 \times 10^{-6}$	$.952 \times 10^{-6}$
1 correction	$.224 \times 10^{-8}$	$.827 \times 10^{-7}$
2 corrections	$.344 \times 10^{-10}$	$.350 \times 10^{-9}$
6th Order Fully Implicit Runge-Kutta based on Lobatto Points [17]		
h = 1/6	$.401 \times 10^{-11}$	$.596 \times 10^{-9}$
8th Order Extrapolation Scheme Based on the Trapezoidal Rule [12]		
h = 1/24	$.401 \times 10^{-11}$	$.109 \times 10^{-10}$
Deferred Correction Method of Pereyra [13]		
h = 1/17	$< 10^{-12}$	$.535 \times 10^{-11}$

Runge–Kutta formula has a cost proportional to $9N^3$ for large N. The deferred correction scheme obtains high accuracy but requires 17 grid points. In fact, to achieve an accuracy of 10^{-9}, which is comparable to the accuracy achieved by the fourth order defect correction algorithm, the deferred correction method still required 17 grid points. This is in line with the conclusions of [7] which found that Runge–Kutta formulae can often achieve low precision solutions using considerably fewer grid points than are required by deferred correction algorithms. This means that the Runge–Kutta formulae can be competitive with deferred correction methods even though the cost of the linear algebra at each point may be higher.

We complete our numerical results for separated boundary conditions by considering three additional problems. Again, these problems are ones which have been considered by other authors and our aim will be to show that Runge–Kutta methods can achieve low accuracy solutions using fewer grid points than deferred correction schemes. The problems considered are

Problem 2

$$\begin{aligned} y_1'(t) &= y_2(t) & y_1(0) &= 1 \\ y_2'(t) &= -y_2(t) - y_1^2(t) + e^{-2t} & y_1(1) &= e^{-1}. \end{aligned}$$

Problem 3

$$\begin{aligned} y_1' &= y_2, & y_1(0) &= 0 \\ y_2' &= y_1^3 - \sin t(1 + \sin^2 t) & y_1(\pi) &= 0. \end{aligned}$$

Problem 4

$$\begin{aligned} y_1' &= y_2 & y_1(0) &= 0 \\ y_2' &= 400(y_1 + \cos^2 \pi t) + 2\pi^2 \cos 2\pi t, & y_1(1) &= 0. \end{aligned}$$

In Table 2 we give the number of grid points required to obtain specified tolerances by the deferred correction method of Lentini and Pereyra[13], the fourth order algorithm 2 of [7] and the 4th order DIRK formula (4.2). As can be seen, both Runge-Kutta algorithms require fewer grid points than *LP* at low orders of accuracy and the

Table 2. Number of grid points required to solve problems 2–4

Problem 2			
Tol	LP	Algorithm 2 of [7]	4th order DIRK
10^{-3}	7	5	5
10^{-6}	13	5	5
10^{-9}	25	5	6
Problem 3			
Tol	LP	Algorithm 2	4th order DIRK
10^{-3}	9	6	5
10^{-6}	17	6	6
10^{-9}	17	11	12
Problem 4			
Tol	LP	Algorithm 2	4th order DIRK
10^{-3}	33	9	11
10^{-6}	33	18	19

performance of Algorithm 2 and the 4th order formula (4.2) is very similar. However, the iteration scheme used with the DIRK formula consistently converged more rapidly than the one used with algorithm 2 of[7].

We shall not give any results for problems with non-separated boundary conditions since the conclusions to be drawn for such problems seem to be broadly the same as in the separated case, i.e. Runge–Kutta formulae can achieve low accuracy solutions using relatively few grid points. Furthermore, there seems to be a marked lack of published results for non-separated boundary conditions with which to compare our methods since many other methods become significantly less efficient when applied to the non-separated, rather than separated, case. We have not considered at all the performance of our algorithms, and particularly methods for increasing the accuracy of the basic Runge–Kutta method, when a variable grid spacing is used. In particular, we need to examine how our algorithms perform when confronted by problems with boundary layers. This is beyond the scope of the present paper but we hope to be able to present results dealing with the variable grid problem in the future.

CONCLUSION

The main purpose of the present paper was to derive an efficient class of Runge–Kutta formulae for the numerical solution of first order systems of two-point boundary value problems. As a consequence of this, we also examined the implicit mid-point rule which is a particular member of the class of Runge–Kutta formulae considered. The theory for Runge–Kutta formulae applied to two-point BVP's has been completely developed by Keller and Weiss and is immediately applicable to our methods. In the case of non-separated boundary conditions we derived an integration method which is significantly cheaper to implement than other related finite difference methods. This approach also carries over to the trapezoidal rule for the solution of linear equations and makes deferred correction a cheaper prospect for this particular class of problems.

In the case of separated boundary conditions we gave an algorithm which is marginally cheaper to implement than conventional implementations. The operation count compares very favourably with those given by Weiss for fully implicit R–K formulae. In Section 5 some results were given using two particular DIRK formulae in a defect correction framework and these methods were found to compare favourably with certain existing methods.

Acknowledgement—It is a pleasure to acknowledge the help of an anonymous referee whose many comments and suggestions greatly improved this paper.

REFERENCES

1. R. Alexander, Diagonally implicit Runge–Kutta methods for stiff O.D.E.s. *SIAM J. Numer. Anal.* **14**, 1006–1021 (1977).
2. T. A. Bickart, An efficient solution process for implicit Runge–Kutta methods. *SIAM J. Numer. Anal.* **14**, 1022–1027 (1977).
3. J. C. Butcher, On the implementation of implicit Runge–Kutta methods. *BIT* **16**, 237–240 (1976).
4. J. C. Butcher, Coefficients for the study of Runge–Kutta integration processes. *J. Austral. Math. Soc.* **3**, 185–201 (1963).
5. J. R. Cash, Diagonally implicit Runge–Kutta formulae with error estimates. *J.IMA* **24**, 293–301 (1979).
6. J. R. Cash, Block Runge–Kutta methods for the numerical integration of initial value problems in O.D.E.s. *Math. Comp.* **40**, 175–206 (1983).
7. J. R. Cash, A variable order deferred correction algorithm for the numerical solution of nonlinear two point boundary value problems, *Comput. Maths. Appls.* **9**, 257–265 (1983).
8. J. R. Cash and A. Singhal, High order methods for the numerical solution of two point boundary value problems. *BIT* **22**, 184–199 (1982).
9. G. J. Cooper and A. Sayfy, Semi-explicit A-stable Runge–Kutta methods, *Math. Comp.* **33**, 541–556 (1979).
10. R. Frank, The method of iterated defect correction and its application of two-point boundary value problems, part 1. *Numer Math.* **25**, 409–419 (1976).
11. R. Frank and C. W. Ueberhuber, Iterated defect correction for differential equations, Part 1. Theoretical results. *Computing* **20**, 207–228 (1978).
12. H. B. Keller, Accurate difference methods for nonlinear two-point boundary value problems. *SIAM J. Numer. Anal.* **11**, 305–320 (1974).
13. M. Lentini and V. Pereyra, An adaptive finite difference solver for nonlinear two-point boundary problems with mild boundary layers, *SIAM J. Numer. Anal.* **14**, 91–111 (1977).

14. S. P. Nørsett, Semi-explicit Runge–Kutta methods. *Mathematics and Computation Report No.* 6/74. University of Trondheim (1974).
15. R. D. Skeel, A theoretical framework for proving accuracy results for deferred corrections. *SIAM J. Numer. Anal.* (1981).
16. J. M. Varah, On the efficient implementation of implicit Runge–Kutta methods. *Math Comput.* **33**, 557–561 (1979).
17. R. Weiss, The application of implicit Runge–Kutta and collocation methods to boundary value problems. *Math. Comput.* **28**, 449–464 (1974).
18. P. E. Zadunaisky, On the estimation of errors propagated in the numerical integration of ordinary differential equations. *Numer. Math.* **27**, 21–39 (1976).